

תרגיל בית #1

מטרות התרגיל:

1. להעמיק את הבנתכם בנושא תהליכים וזימון תהליכים ב-Linux.
2. הכרות בסיסית עם קריאות מערכת אלמנטריות.
3. הבנה של נושא האיתותים ב-Linux.



תרגיל רטוב: כתיבת smash

שימו לב: מקוריות הקוד תיבדק, להזכירכם העתקת שיעורי בית הינה עבירה משמעת בטכניון לכלל המשתמע מכך.

עליכם לכתוב תוכנית אשר תשמש כ-shell חדש למערכת ההפעלה Linux. התוכנית תבצע פקודות שונות אשר יוקלדו ע"י משתמש.

השם smash נגזר מצרורף המילים **Small Shell**.

התוכנית תפעל בצורה הבאה:

- התוכנית ממתינה לפקודות אשר יוקלדו ע"י המשתמש ומבצעת אותן (וחוזר חלילה).
- התוכנית תוכל לבצע מספר קטן של פקודות built-in, הפקודות יפורטו בהמשך.
- כאשר התוכנית תקבל פקודה שהיא לא אחת מפקודות ה-built-in היא תנסה להפעיל אותה כמו shell רגיל, אופן הפעלת פקודה חיצונית יפורט בהמשך.
- **במידה והוכנסה פקודת built-in עם פרמטרים לא חוקיים או כמות פרמטרים לא נכונה, תופיע הודעת השגיאה הבאה:**

smash error: > "line"

כאשר:

- **הגרשיים יופיעו בהודעת השגיאה.**
 - **line היא שורת הפקודה כפי שהוקשה על ידי המשתמש.**
 - **על השגיאות שפורטו ויפורטו בהמשך, התוכנית מגיבה בהדפסת הודעת שגיאה מתאימה ועוברת לפענוח וביצוע שורת הפקודה הבאה.**
 - כאשר התוכנית ממתינה לקלט מהמשתמש מודפסת בתחילת שורה חדשה ההודעה¹
- smash >**
- **ניתן להשתמש בכל מספר רווחים בין מילים המופיעות באותה שורת פקודה, ובתחילת השורה.**
 - כל פקודה נגמרת בתו '\n'.
 - ניתן להשאיר שורות ריקות.
 - כל פקודה מופיעה בשורה נפרדת ואורכה לא יעלה על 80 תווים.
 - ניתן להניח שמספר הארגומנטים המקסימאלי הינו 20.

אופן פענוח שורת פקודה ב smash:

כפי שיפורט בהמשך התוכנית תבדוק אם הפקודה היא פקודת built-in או פקודה חיצונית, ותטפל בפקודה בהתאם.

פקודות חיצוניות ב-smash:

<command> [arguments]

כאשר smash מקבלת פקודה חיצונית (כלומר אינה אחת מהפקודות built-in של smash) היא מנסה להפעיל את התוכנית command. וממתינה עד לסיום ביצוע התוכנית. לדוגמא, הפקודה:

smash > a.out arg1 arg2

תגרום להפעלת התוכנית a.out עם הארגומנטים arg1 arg2.

אם פעולת הפעלת התוכנית החיצונית לא הצליחה, תודפס הערת שגיאה המתארת את סיבת כישלון הפעולה (תזכורת – perror).

¹ ההודעה "smash >" ידועה גם בשמה הטכני - prompt.

& [arguments] <command>

כמו בסעיף הקודם, אך ללא המתנה לסיום ביצוע התוכנית (הרצה ברקע). התהליך החדש יכנס לרשימת jobs (ראה פקודת jobs ברשימת פקודת ה-built-in).

פקודות built-in של smash :pwd

הדפס את מיקומו של המדריך הנוכחי.

cd <path>

שנה את המדריך הנוכחי ל- path. אם ה- path אינו נכון הדפס הודעת שגיאה.

smash error: > No such file or directory

במקרה בו path שווה ל "-", משנים את המדריך הנוכחי אל הקודם ומדפיס אותו (אם קיים). צריך לזכור רק מדריך אחד אחורה. לדוגמא:

```
smash > pwd
/foo
smash > cd /bar
smash > pwd
/bar
smash > cd -
/foo
smash > cd -
/bar
smash > cd -
/foo
```

history

מדפיס למסך את היסטוריית הפקודות של smash, כל פקודה בשורה חדשה, יחד עם הפרמטרים, כאשר הפקודה שהורצה אחרונה הינה מודפסת למסך אחרונה. לדוגמא:

```
smash > history
pwd
pwd
cd -
cd -
cd -
```

אם לא הורצו פעולות, הפקודה לא תדפיס כלום. הפקודה תשמור עד 50 פקודות אחורה ותמחק את הרשומות הכי ישנות כאשר אין מקום ע"מ לפנות מקום עבור פקודה חדשה. אין צורך להכניס את הפקודה history להיסטוריה.

jobs

ה- smash יחזיק רשימת jobs, את pidn ואת הזמן בשניות מהרגע שהתהליך נכנס לרשימת jobs של כל אחד מהם. ברשימה יופיעו כל התהליכים שהופעלו ברקע (ע"י &) אך טרם הסתיימו. בנוסף, יופיעו כל התהליכים שנעצרו ע"י סיגנל מתאים, ולידם יופיע התיאור Stopped. הפקודה תציג רשימת jobs וpids, מספר סידורי לפני לכל תהליך. מספר סידורי הוא מספר שלם מונוטוני עולה ממש. לדוגמא:

```
smash > jobs
[1] a.out : 12340 214 secs
[2] /usr/bin/ls: 12341 57 secs
[3] b.out : 12342 10 secs (Stopped)
```

אין צורך לעשות שימוש חוזר במספרי jobs. למשל, אם תהליך מספר 2 הסתיים ויצא מהרשימה, התהליך הבא שיכנס לרשימה יכול להיות עם מספר 4. שמות התהליכים יכולים להיות עם או בלי הפרמטרים שהועברו לתוכנית.

kill -<signum> <job>

שליחת Signal שמספרו signum אל התהליך המזדהה עם **job** (מרשימת ה-jobs). במידה וjob אינו קיים יש להדפיס :

smash error: > kill job – job does not exist

במידה ויש כישלון אחר בשליחת ה-Signal יש להדפיס :

smash error: > kill job – cannot send signal

הערה : job הינו מס' job ברשימת ה-jobs ולא pid. יש "לפני מספר ה Signal".

showpid

ה-smash ידפיס את הpid שלו (יודפס ה-pid של ה-smash). לדוגמא :

smash > showpid

smash pid is 12339

fg [command number]

הפקודה תגרום להרצה ב- foreground של התהליך (job) המזוהה עם *command number*. לפני העברת ה-job ל- foreground יודפס שמו. הפעלת הפקודה ללא פרמטרים, תעביר ל- foreground את התהליך האחרון (כלומר, התהליך עם job id מקסימלי) שהופעל ברקע. אם התהליך במצב Stopped, יש להמשיך את ריצתו ב- foreground. **כשהתהליך יסתיים הוא יוצא מרשימת jobs.**

לדוגמא : (המשך לדוגמא מסעיף הקודם)

smash > fg

b.out

smash > jobs

[1] a.out : 12341 218 secs

[2] /usr/bin/ls : 12342 61 secs

smash > fg 1

a.out

bg [command number]

הפקודה תגרום להרצה ב- background של התהליך (job) המזוהה עם *command number*. אם התהליך לא במצב Stopped תוצג הודעת שגיאה מתאימה. לפני העברת ה-job ל- background יודפס שמו. הפעלת הפקודה ללא פרמטרים, תעביר ל- background את התהליך האחרון (כלומר, התהליך עם job id מקסימלי) שריצתו הושהתה (ע"י סיגנל מתאים). **כשהתהליך יסתיים הוא יוצא מרשימת jobs.**

שימו לב: התהליך ירוץ ברקע, כלומר על ה-smash לא לחכות לסיום התהליך אלא להחזיר את הprompt מיידית. לדוגמא :

smash > jobs

[1] a.out : 12340 56 secs

[2] /usr/bin/less : 12341 23 secs (Stopped)

[3] c.out : 12342 10 secs (Stopped)

smash > bg

c.out

smash > bg 2

/usr/bin/less

quit [kill]**א. quit**

יציאה מתוכנית ה-smash.

ב. quit kill

הרחבה לפקודת ה-quit היא לאפשר למשתמש להרוג את כל התהליכים בעת היציאה.

הפקודה quit תהרוג את התהליכים לפי האלגוריתם הבא :

1. שליחת סיגנל SIGTERM.

2. (רק) אם התהליך לא נהרג אחרי 5 שניות לאחר קבלת סיגנל ה-SIGTERM, שליחת

סיגנל SIGKILL.

הערה: אם ברצונכם לבדוק אופן זה של quit, ניתן לייצר תוכנית דמה אשר מתעלמת מסיגנל ה-

SIGTERM.

לדוגמה :

smash > jobs

```
[1] a.out 12340 56 secs
[2] /usr/bin/ls 12341 23 secs
[3] b.out 12342 10 secs
```

smash > quit kill

[1] a.out – Sending SIGTERM... Done.

[2] /usr/bin/ls – Sending SIGTERM... Done.

[3] b.out – Sending SIGTERM... (5 sec passed) Sending SIGKILL... Done.

הערה: תהליך b.out לא הגיב לסיגנל SIGTERM, ולכן נשלח לו גם SIGKILL.

cp <old name> <new name>

מעתיק את הקובץ old_name לnew_name. אפשר להניח כי old_name הינו קובץ ולא תיקייה.

לאחר **העתקה** מוצלחת יודפס למסך "<new_name> has been copied to <old_name>". אם

הייתה תקלה (לדוגמה, old_name לא קיים) יש להדפיס אותה באמצעות perror.

לדוגמא :

smash > cp a.out b.out

a.out has been copied to b.out

diff <f1> <f2>

הפונקציה משווה את תוכן הקבצים f1 ו-f2. אפשר להניח כי f1 ו-f2 הינם קבצים ולא תיקיות.

הפונקציה תדפיס למסך "1" אם תוכן הקבצים שונה, ואחרת תדפיס "0". אם הייתה תקלה

(לדוגמה, אם f1 או f2 לא קיימים). יש להדפיס אותה באמצעות perror.

לדוגמא :

smash > diff a.out b.out

1

עליכם לממש את הפונקציות ה- built in ולא להשתמש בfork+exec עבורן. יש לקרוא ל- System Calls המתאימות. חלקן לא נלמדו בכיתה ולכן יש להשתמש בספר הקורס או ב-man/google על מנת למצוא את התיעוד של קריאות המערכת המתאימות.

הצגת signals ב-smash:

כל פעם ששולחים signal pid כלשהו, ה-smash יציג את pid וסוג הסיגנל שנשלח. לדוגמא, אם fg צריך להעיר תהליך מושהה:

```
smash > fg 1
smash > signal SIGCONT was sent to pid 12340
```

השהיית/הריגת התהליך:

על ה-shell לתמוך בצירופי המקשים CTRL+C ו-CTRL+Z:

- הצירוף CTRL+Z משהה את התהליך שרץ ב-foreground (שולח לו SIGTSTP) ומוסיף אותו לרשימת ה-jobs (עם ציון שהתהליך מושהה) לדוגמא:

```
smash > jobs
[1] a.out 12340 23 secs (Stopped)
[2] /usr/bin/ls 12341 10 secs
```

- השהיית תהליך יכולה להתבצע גם באמצעות kill עם סיגנל מתאים.
- לאחר השהיית התהליך, הקשת הפקודה fg תגרום לשחזור הריצה של התהליך המושהה ב-foreground (ע"י שליחת SIGCONT). בנוסף, יש לתמוך בפקודה bg אשר תגרום לשחזור הריצה של התהליך המושהה ב-background.
- הצירוף CTRL+C מפסיק את ריצת התהליך שרץ ב-foreground (שולח SIGINT).

שימו לב 1: אם אין פקודה ב-foreground, צירופים אלו לא ישפיעו על ה-shell.

שימו לב 2: ה-shell שלכם נדרש רק לנתב את הסיגנל לתהליך שרץ ב-foreground של ה-shell.

שימו לב 3: כאשר אתם מריצים תוכנית באמצעות exec כל ה-signal handlers חוזרים לdefault.

שימו לב 4: אתם עלולים לגלות כי גם תהליך ה-smash וגם כל תהליכי הבן שלו מקבלים את הסיגנלים CTRL+C ו-CTRL+Z למרות שה-smash שלכם לא שולח אותם לתהליך הבן! בעיה זו מתרחשת בגלל ה-shell האמיתי (tcsh, bash, ...). שממנו רץ ה-shell שלכם, אשר שולח את הסיגנל לכל התהליכים בעלי אותו ה-group-id. בכדי להימנע מבעיה זאת אתם פשוט צריכים לשנות את ה-group-id של כל תהליך בן אשר ה-shell שלכם מייצר באופן הבא:

```
// here your code runs the child process
pid = fork();
if( pid == 0 ) {
    setpgrp(); // THIS IS THE COMMAND THAT EACH CHILD SHOULD
               // EXECUTE, IT CHANGES THE GROUP ID
    execv(...); // execute the needed process
    // Handle execv error if you got to this line
} else if( pid > 0 ) {
    // Do parent - shell work here
} else {
    // handle the fork error here
}

שימו לב 5: כעת כאשר נלחצים הצרופים CTRL+C ו-CTRL+Z הסיגנלים מנותבים ע"י ה-shell המקורי ל-smash (ולא לתהליך הרץ בחזית).


שימו לב 6: עליכם לתפוס את הסיגנלים ב-smash ולנתב אותם לתהליך שרץ בחזית.


```

הנחיות לביצוע

- יש להשתמש בקריאות המערכת fork ו-exec (יש לבחור את הצורה המתאימה של exec לדרישות התרגיל).
- אין להשתמש בפונקציית הספרייה system.
- על התוכנית לבדוק הצלחת ביצוע כל פקודה, בכל מקרה של כישלון יש להדפיס הודעת שגיאה מתאימה (תזכורת – perror).
- יש לממש את התרגיל ב-C או C++ בלבד.
- ניתן להשתמש בספריות STL של C++ בחופשיות, ולכן מומלץ לכתוב את התרגיל ב-C++.
- על מנת להמנע מכתובת מבני נתונים ב-C.

הידור קישור ובדיקה

יש לוודא שהקוד שלכם מתקמפל ע"י הפקודה הבאה:
אם כתבתם ב-C++:

```
> g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG *.cpp -o smash
```

אם כתבתם ב-C:

```
> gcc -std=c99 -Wall -Werror -pedantic-errors -DNDEBUG *.c -o smash
```

יש לוודא שנוצר קובץ הרצה ללא שגיאות או warnings.

עליכם לספק Makefile עבור בניית הקוד. הכללים המינימליים שצריכים להופיע ב-Makefile הינם:

- כלל smash שיבנה את התוכנית smash.
- כלל עבור כל קובץ נפרד שקיים בפרויקט.
- כלל clean אשר מוחק את כל תוצרי הקימפול.
- יש לוודא שהתוכנית נבנית ע"י הפקודה make.
- יש לקמפל ע"י הדגלים המופיעים בחלק "הידור קישור ובדיקה" לעיל.

לתרגיל זה מצורף סקריפט check_submission.py המוודא (בצורה חלקית) את תקינות ההגשה. הסקריפט מצורף לנוחיותכם, ובנוסף לבדיקה באמצעות הסקריפט, עליכם לוודא את תקינות ההגשה.

הסקריפט מצפה ל-2 פרמטרים: נתיב ל-zip, ושם קובץ ההרצה. לדוגמא:
> ./check_submission.py 123456789_987654321.zip smash

הגשה

הנחיות כלליות על אופן הגשת תרגילי הבית הרטובים ניתן למצוא באתר הקורס תחת הכותרת "עבודות בית – מידע ונהלים".

- אנא עקבו אחר ההנחיות המופיעות בדף הנהלים. יש להגיש קובץ zip (ולא אף פורמט אחר) בלבד.
- אין להגיש קבצי הרצה.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.
- על ה-Makefile המצורף לייצר קובץ הרצה בשם "smash".

**בבקשה, בדקו שהתוכניות שלכם עוברות קומפילציה
וההגשה נעשית על פי הנהלים.
תוכנית שלא תעבור קומפילציה לא תבדק!
הגשה שלא על פי הנהלים תגרור הורדת ציון.**

Useful Man Pages (non-exhaustive list):

`exec(3),fork(2),wait(2),waitpid(2),pause(2),signal(2) or sigaction(2),
stat(2),open(2),read(2),write(2),close(2)`

בהצלחה!!!